Making a Fortran program for solving a heat equation using the forward-difference (Euler) method

Written by Evgenii Neumerzhitckii

Aug 23, 2019

Contents

The heat equation

We want to use a finite-difference (Euler) method to find an approximate solution of the heat equation

$$T_t = kT_{xx} \tag{1}$$

with the initial condition

$$T(x, 0) = 100 \sin(\pi x/L), \tag{2}$$

and boundary conditions

$$T(0, t) = T(L, t) = 0 \tag{3}$$

for $L = 1\ m$. Here we use a short notation for the derivatives: $T_t = \frac{\partial T}{\partial t}$.

The recurrence relation

Our goal is to find a recurrence relation that will allow us to calculate the value of temperature iteratively at each time step. In order to find this recurrence relation we use approximations for the derivatives. The time derivative approximation is

$$T_t(x, t) \approx \frac{T(x, t + \Delta t) - T(x, t)}{\Delta t},$$

with the corresponding approximate expression

$$(T_t)_j^n = \frac{T_j^{n+1} - T_j^n}{\Delta t}, \tag{4}$$

where the j is the position index and n is the time index:

$$j = 1, 2, \ldots, n_x$$

$$n = 1, 2, \ldots, n_t,$$

with n_x and n_t being the total number of position and time values respectively. Similarly, the approximate expression for the second position derivative is:

$$(T_{xx})_j^n = \frac{T_{j+1}^n - 2T_j^n + 2T_{j-1}^n}{\Delta x^2}. \tag{5}$$

Next, we substitute Equations 4 and 5 into Equation 1:

$$\frac{T_j^{n+1} - T_j^n}{\Delta t} = k\frac{T_{j+1}^n - 2T_j^n + 2T_{j-1}^n}{\Delta x^2}.$$

Finally, we solve for T_j^{n+1} and find the recurrence relation that we wanted

$$\boxed{T_j^{n+1} = T_j^n + \alpha\left(T_{j+1}^n - 2T_j^n + 2T_{j-1}^n\right), \quad \alpha = k\frac{\Delta t}{\Delta x^2}.} \tag{6}$$

Writing the code

Next, we write Fortran code to solve the heat equation (Equation 1). The code is shown in shown in Listing 1.

Listing 1: Solving a heat equation with forward-difference method (`heat_equation.f90`).

```
77 ! Assign evenly spaced x values
78 call linspace(x0, x1, x_points)
79
80 ! Set initial conditions
81 data(:, 1) = 100._dp * sin(pi * x_points / l)
82
83 ! Set boundary conditions
84 data(1, :) = 0
85 data(nx, :) = 0
86
87 ! Calculate numerical solution using forward differencing method
88 do n = 1, nt - 1
89     data(2:nx-1, n + 1) = data(2:nx-1, n) + &
90         alpha * (data(3:nx, n) - 2 * data(2:nx-1, n) + data(1:nx-2, n))
91 end do
```

On `Line 77` we assign evenly spaced values between 0 and 1 for the x-coordinate and store then in the `x_points` array. The number of the values nx are supplied to the program by the user.

Next, on `Line 81` use the initial condition from Equation 2. Our temperature values are stored in a 2D array variable called `data`. Here we assign the temperatures to all the x-values corresponding to the first time index $n = 1$.

Similarly, on `Line 77` and `78` we use the boundary conditions (Equation 3) by assigning zero temperatures to the ends of the rod. This is done by assigning zero for all time indexes in the `data` array corresponding to first (1) and last (nx) x-index.

Finally, on `Lines 88-91` we iterate over time indexes (n) and use the recurrence relation from Equation 6 to assign the temperature value for the next time index (`n + 1`) using the temperatures that were calculated for the previous step for the time index (n).

On `Lines 89-90` are using a so-called "vectorized" indexing syntax for the x-coordinate, such as `2:nx-1`. This allows the program to use SIMD processor instructions that perform multiple operations in one cycle. These instructions take advantage of the fact that x-values are located contiguously in memory (Fortran arrays are column-major, meaning the values from the first index of an array are stored one after another in memory). This index notation will make out calculation multiple times faster (this speed increase depends on type of SIMD instructions implemented in a specific processor) compared to an alternative implementation where we would iterate over the x values using a loop.

Running the program and making plots

Instructions for compiling, running the program and plotting its results are located in the README.md file that comes with the source code.

Approximate solution and errors

We look at the approximate solution produced by our program by first plotting the solution for small number of x values ($nx = 5$) on Figure 1.

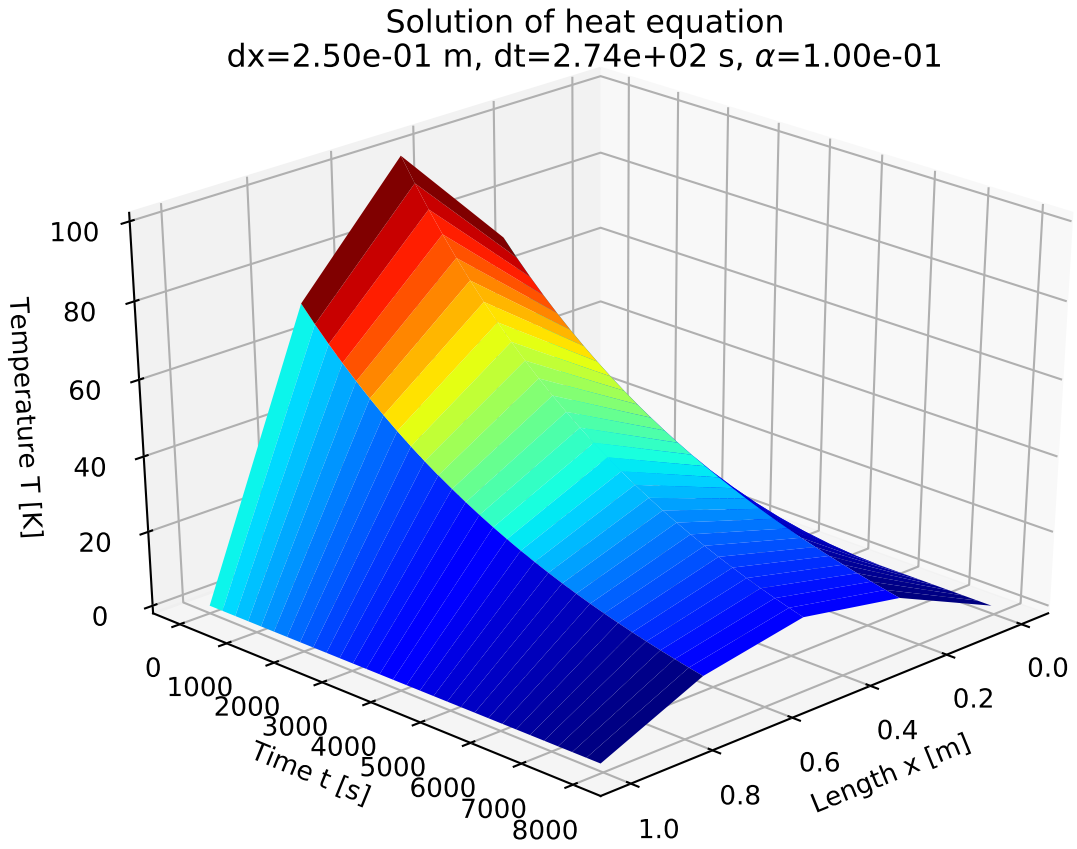

Figure 1: Approximate solution to the heat equation for $nx = 5$ and $\alpha = 0.1$.

We can see from Figure 1 that the the temperature of the rod decreases from a sinusoidal distribution with time, and the ends remain at $T = 0$, as expected.

The corresponding errors are shown on Figure 2. The errors were found by calculating the absolute value of the difference between the approximate solution and the exact solution, which is given by equation

$$T(x,t) = 100e^{-\pi^2 kt/L^2} \sin(\pi x/L).$$

We can see from Figure 2 that the errors are increasing with time, and that the errors are larger near the center of the rod. We can see that the errors reach peak values of about $0.6K$ at $t \approx 3000\ s$.

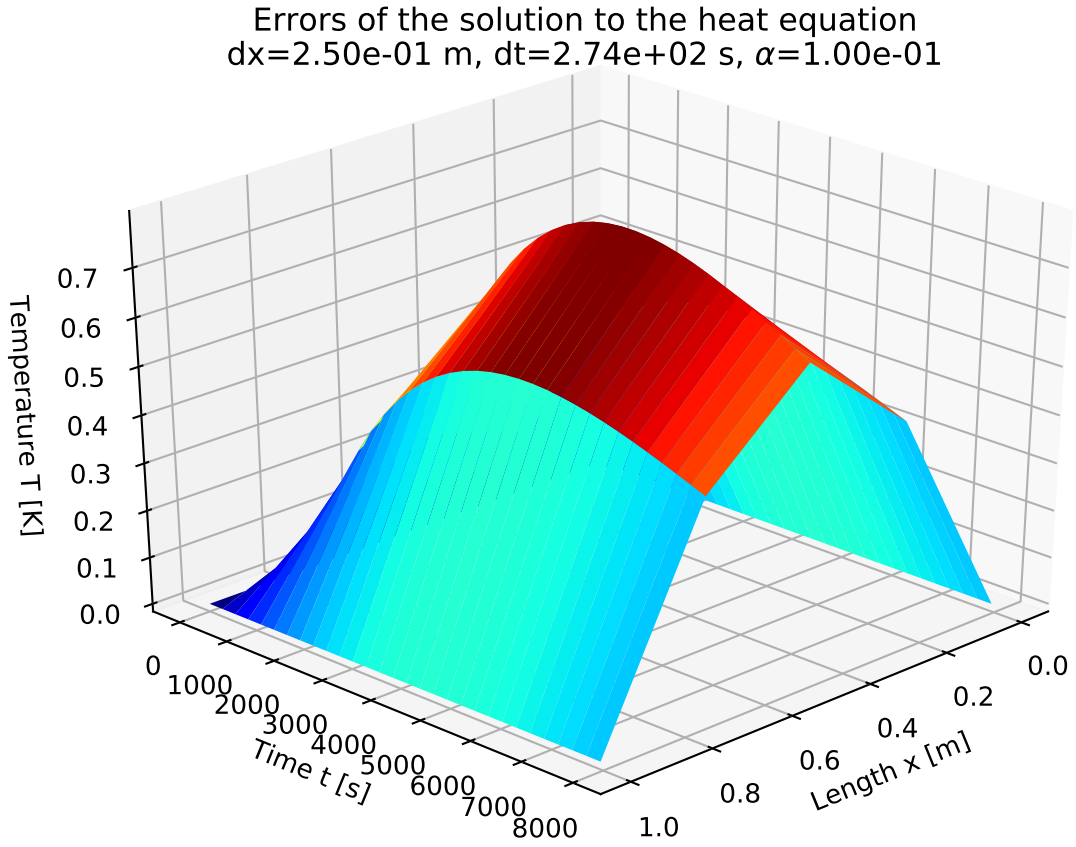

Figure 2: The absolute errors of the approximate solution of the heat equation for $nx = 5$ and $\alpha = 0.1$.

Solution for 21 position steps

Next, we use larger amount of position steps, $nx = 21$ (we are counting both edges of the rod) and $\alpha = 0.25$. The solution is shown on Figure 3. We have calculated the solution for 300 time steps ($nt = 300$).

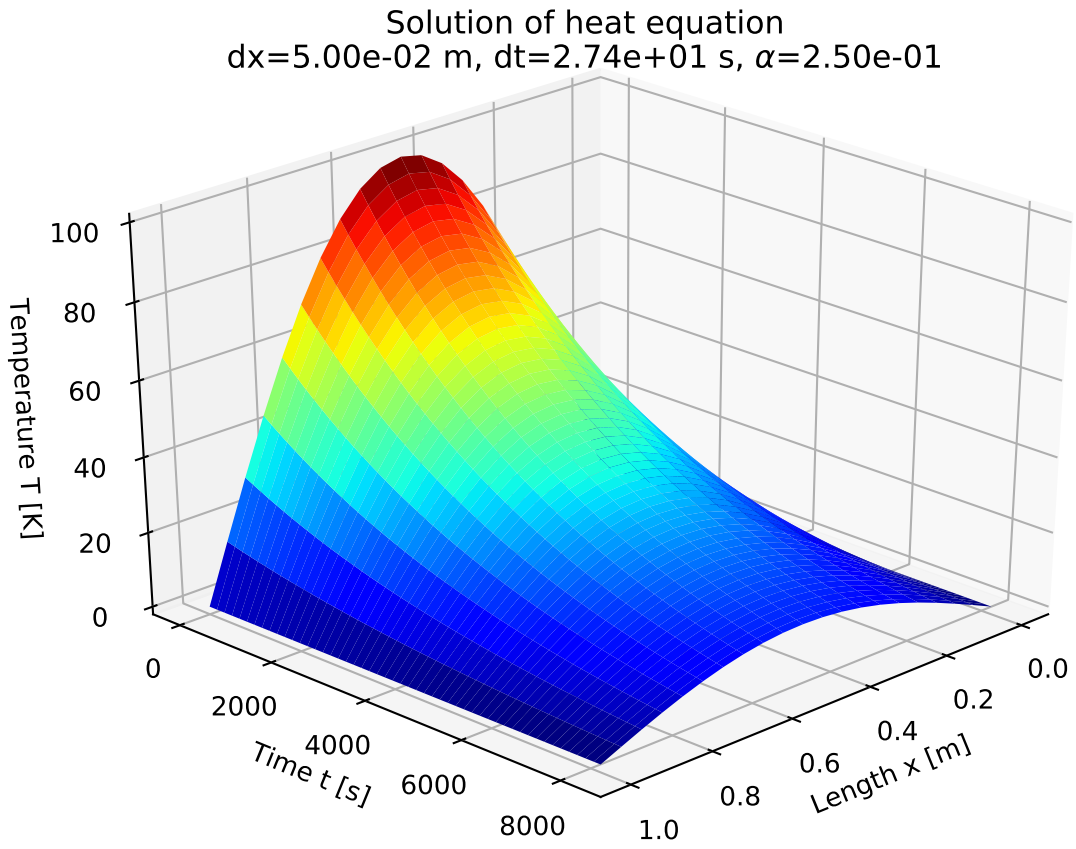

Figure 3: Approximate solution to the heat equation for $nx = 21$ and $\alpha = 0.25$.

The general shape of the solution is not unlike the one we found for $nx = 5$. We can see that the new $nx = 21$ solution is smoother, which can be explained by smaller steps for position and time. The errors of the approximate solution for $nx = 21$ are shown on Figure 4.

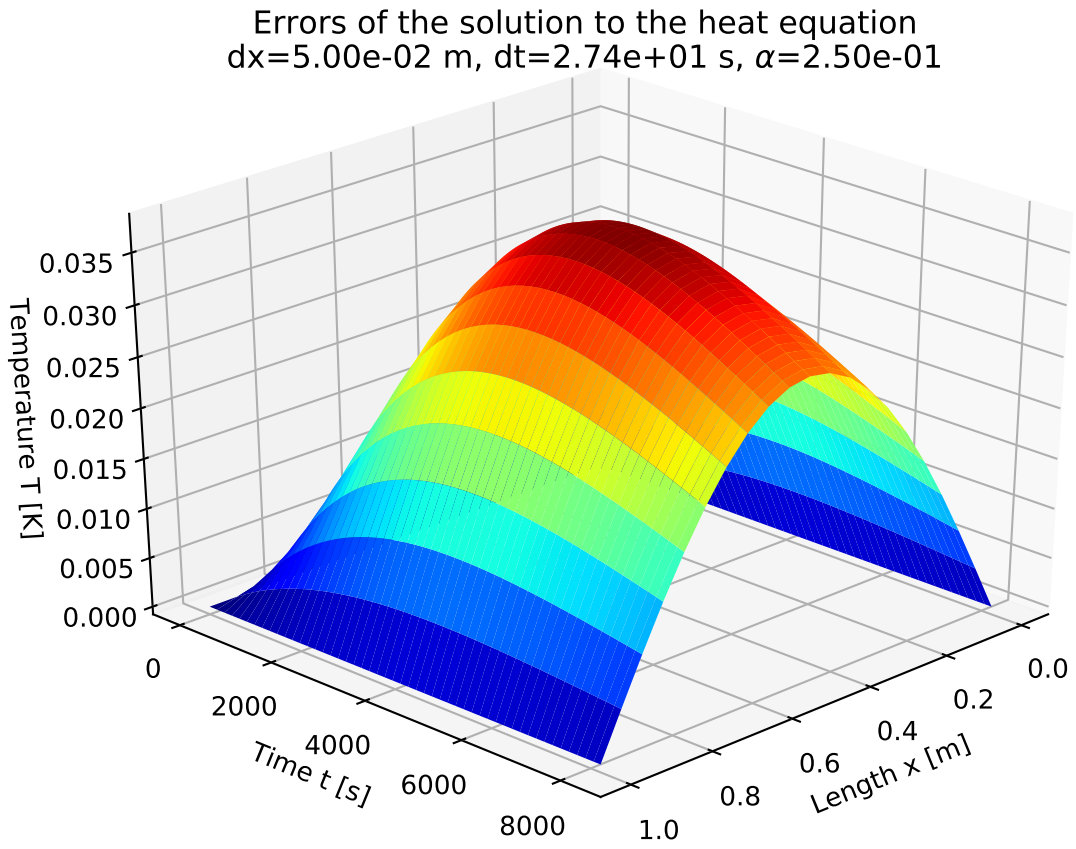

Figure 4: The errors of the approximate solution to the heat equation for $nx = 21$ and $\alpha = 0.25$.

We can see similar distribution of errors as before. This time, however, the peak values of errors are about 20 times smaller, reaching maximum values of about 0.03 K.

Solution for $\alpha = 0.65$

Here we want to see how values of α parameter affect the solution. Specifically, we use $\alpha = 0.65$ while keeping the same number of position steps ($nx = 21$). The solution is shown on Figure 5.

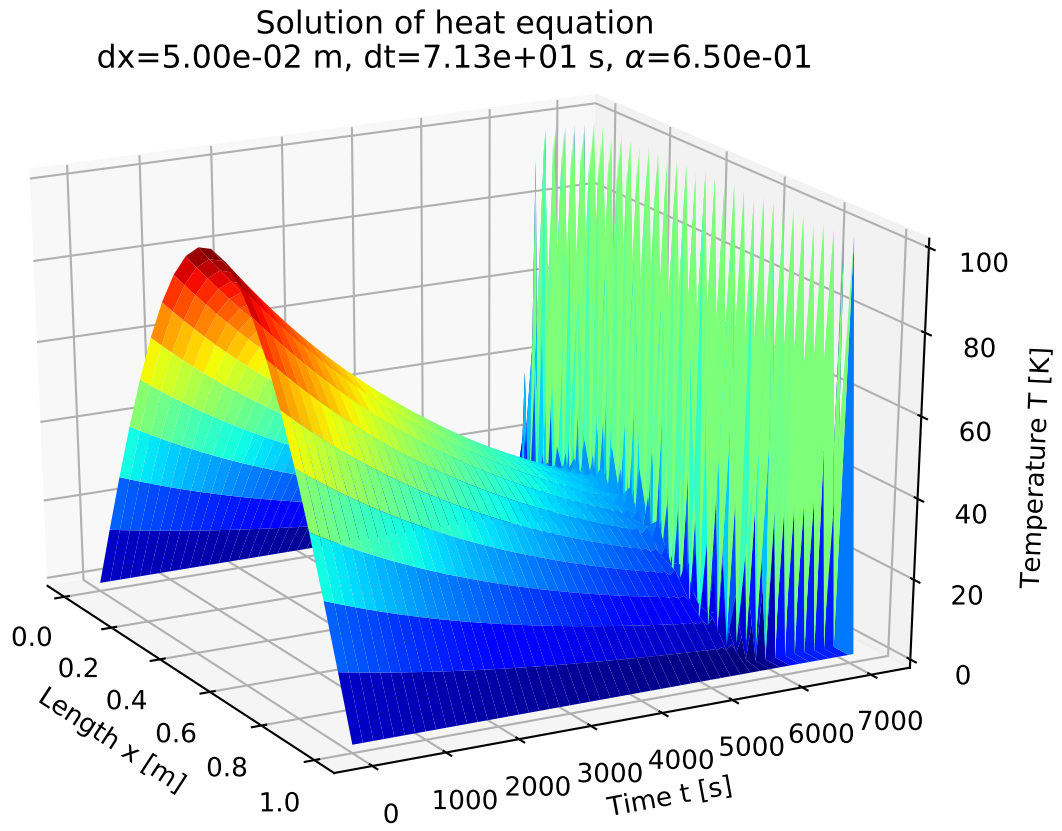

Figure 5: Approximate solution to the heat equation for $nx = 21$ and $\alpha = 0.65$, showing signs of numerical instability from $t \approx 5500\ s$.

We can see form Figure 5 that initially, the program produces results similar to the previous case with $\alpha = 0.25$. However, after about $t \approx 5500\ s$, the $\alpha = 0.65$ solution shows large fluctuations in temperature. These result are unrealistic, since the rood cools down with time, and it can not heat up in random locations without an external supply of energy.

These problems in our numerical solution are likely to come from unbounded growth of rounding errors. A more detailed stability analysis using Von Neumann method is needed to show that the forward-difference method of solving Equation 1 is unstable for $\alpha > 0.5$. This condition limits our choice of the position steps: we can not make Δx as small as we like for same values of Δt. Fortunately, there are alternative methods of solving Equation 1, such as backward-difference and Crank-Nicolson methods, that are always stable for any choice of Δx and Δt.

Conclusion

We used forward-difference (Euler) method and solved a heat equation with initial and boundary conditions. We have found that smaller position and time steps result in smaller errors.

However, we have also found that making our position step too small results in unlimited growth of errors.